

# Infrastructure for Open-Domain Information Extraction

Mihai Surdeanu  
Language Computer Corporation  
6440 North Central Expressway  
Dallas TX 75206  
mihai@languagecomputer.com

Sanda M. Harabagiu  
Language Computer Corporation  
6440 North Central Expressway  
Dallas TX 75206  
sanda@languagecomputer.com

## ABSTRACT

The problem of performing open-domain Information Extraction (IE) was historically tied to the problem of ad-hoc acquisition of extraction patterns. In this paper we show that this requirement is not sufficient and that we also need to build new IE architectures that combine the role of linguistic patterns with coreference knowledge and ambiguous syntactic and semantic information. We present the implementation of a novel IE architecture, namely the CICERO system and show how (1) both high precision and high recall results were obtained for a variety of extraction domains; and (2) how textual information can be extracted for virtually any domain in a precise and reliable way. The evaluation of CICERO's performance shows a significant improvement over MUC IE systems.

## 1. INTRODUCTION

In the 90s, the Message Understanding Conferences (MUCs) and the TIPSTER programs gave great impetus to research in information extraction (IE). The systems that participated in the MUCs have been quite successful at extracting information from newswire messages and filling templates with the information pertaining to the events of interest. Typically, the templates model queries regarding *who* did *what* to *whom*, *when* and *where*, and eventually *why*. The most remarkable results concerned the performance of some of the IE subtasks that has reached near-human precision. For example, current Named Entities were recognized with over 90% precision. Not as successful was the broader task of event recognition (i.e. filling in scenario templates) which resulted in at most 60% recall and 70% precision. For a while, this was believed to be a de-facto performance barrier of IE systems. The rationale of this belief is partly explained by the converging results of different IE systems performing scenario template extractions. The other explanation stems from the similar architectures of IE systems that participated in the last MUC evaluations.

In today's world the need for Information Extraction is more pervasive than ever. In fact, we need to be able to operate open-domain IE, in which the domain of interest results from several interactions with the user, in quest of capturing novel data trends from massive

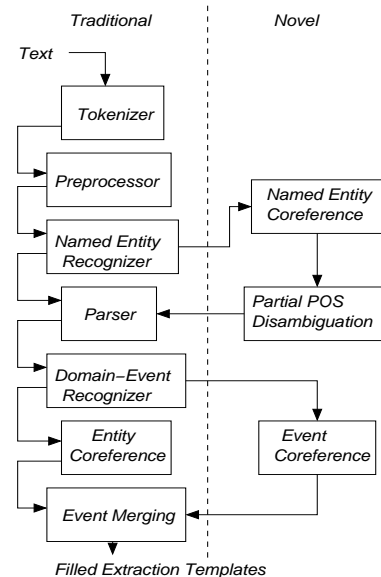


Figure 1: The architecture of the CICERO IE system.

text collections. Successful open-domain IE cannot be achieved only by automatically learning linguistic patterns that encode the domain knowledge. We also need to reliably recognize reference to the same entities or events of interest and we need to disambiguate precisely syntactic and semantic information pertaining to the topic of interest, regardless of domain. Moreover, in this paper we claim that open-domain IE systems need to extract scenario templates with better performance than the typical systems evaluated in MUC. For this reason we have developed the infrastructure that proved to surpass the 70% F-score ceiling, and thus provide high-performance IE for any possible domain. This infrastructure is best illustrated by the architecture of CICERO, the IE system developed by Language Computer Corporation and represented in Figure 1. In order to emphasize the novelties introduced by our information extraction system, we separated CICERO's modules in two categories: modules seen in traditional IE systems, and novel modules introduced in CICERO.

CICERO's first module is the *tokenizer*, whose task is to break the message into lexical entities or tokens. A lexical recognizer identifies tokens or groups of tokens that are entries in dictionaries or gazetteers<sup>1</sup>. The *preprocessor* patterns identify simple numerical lexical entities, such as *money*, *percents*, *dates*, or *times*. The next

<sup>1</sup>Gazetteers are long lists of location or person names.

module recognizes named-entities like *persons*, *organizations*, or *locations*. The *named-entity coreference* is the first novel module introduced by CICERO. Its task is to help disambiguate incomplete or ambiguous named entities. For example, given the name *Austin*, it is classified either as a *person* or a *location* depending on possible reference to previous entities from the text. This module is described in more detail in the next section. The *partial part of speech (POS) disambiguation* module is another CICERO novelty. We use this module to enhance the precision of POS tagging. This feature is important for the resolution of nominal anaphorae in texts, which in turn greatly influences the overall performance of IE, as we will show throughout this paper. We implemented the POS disambiguation module using voting between our lexicons and an external POS tagger (Eric Brill’s POS tagger [2]). The next module is a *phrasal parser*, used to identify noun, verb and particle phrases. As reported in [4], successful IE systems can be implemented by relying only on phrasal parsers and by cascading Finite-State Automata (FSAs) that recognize patterns corresponding to (1) named entities; (2) simple and complex phrases and (3) information relevant to the domain of interest. Such traditional IE architectures (cf. [1], [3], or [5]) prefer finite-state models of language processing over full natural language processing because they would rather do only the *right* language processing for IE instead of deep NLP. This decision was largely inspired by the tremendous success in MUC-3 that the group at the University of Massachusetts obtained with a fairly simple system [6].

Anaphoric expressions hinder the process of matching event patterns. To resolve pronominal and nominal anaphorae, we use an *open-domain coreference* resolution module that generates coreference links between each noun phrase and antecedents that agree in number, gender and are semantically consistent with the anaphora. Coreference resolution enables the *domain event recognizer* to match text against event patterns and to populate the corresponding domain templates/templettes<sup>2</sup>. The task of the *domain coreference* module is to fill template or templette slots that could not be filled using domain patterns. The *merging* module is used to combine templates/templettes that refer to the same event. We implemented merging as a two step process: we first attempt to merge templates found inside the same sentence, because the likelihood of having same-event patterns is higher within one sentence. The second step merges all the templates reported from the first step.

The novel IE architecture implemented in CICERO determined the impressive extraction results it achieves. For example, on the “*management succession*” domain tested in the MUC-6 evaluations, it obtains an F-score of 83%. The F-score is defined as  $F = \frac{P \times R}{R + P}$  where  $P$  is precision and  $R$  is recall.

## 2. IMPROVING EXTRACTION RESULTS

After the MUC-6 evaluations it was believed that the performance of an IE system is directly proportional with the number of rules available. The ACCELERATE program analyzed this claim and showed that it is not valid. Thus, before creating any open-domain IE infra-structure, we need to analyze in depth the conditions that enable the best performing IE system. We have found that four factors are determinant for obtaining high-performance IE:

<sup>2</sup>For each extraction task a domain template is defined, as an interface to database entries. The TIPSTER Message Understanding Conference (MUC) evaluations used predefined templates listed in [?] and [?]. The population of templates required cumbersome rules for filling and scoring. In an attempt to simplify the template processing, the Hub-4 Event99 (cf. [?]) introduced the “templettes”, consisting of just a few slots representing basic information such as main event participants, event outcome, time and location.

1. *Coreference* plays an important role. In fact coreference resolution is an engineering bottleneck in IE systems because (1) it influences *merging* of information referring to same entity or event, and (2) it participates in the *disambiguation* of incompletely defined entities/events. Consider the following text from the “natural disasters” domain:

... flooding has become a way of life in Guerneville in the last several years. This gas station along the Russian River gets hit nearly every time. ...

IE rules enable the recognition of “flooding” and “This gas station along the Russian River gets hit” into partially-filled templettes, e.g.:

```
<NATURAL_DISASTER> :=
  DISASTER: "flooding"
```

```
<NATURAL_DISASTER> :=
  AMOUNT_DAMAGE: "this gas station"
  LOCATION: "Russian River"
```

but it is the coreference module that enables the merging of the two fragments into a single event templette, by linking the two templettes through an event coreference chain:

```
<NATURAL_DISASTER> :=
  DISASTER: "flooding"
  AMOUNT_DAMAGE: "this gas station"
  LOCATION: "Russian River"
```

Coreference chains are used to disambiguate incompletely defined entities, generating a significant increase in the F-measure of our named-entity recognizer. For example, the named-entity “Michigan” can be either a location name, or an organization abbreviation. A coreference chain linking this entity to “Michigan Corp.” disambiguates this entity to a not-so-intuitive organization name.

2. There are *different forms of coreference* that should be processed prior to merging. First coreference is used by the named-entity recognizer to disambiguate ambiguous entities. Example 1 (Mihai). To evaluate the impact of coreference on the named entity recognizer (NER) we evaluated the performance of CICERO’s NER on the MUC-6 test data, with and without coreference help<sup>3</sup>. Our experiments indicated that, for the NE task alone, the coreference module yields over 5% increase in F-measure: the NER F-measure before coreference is used was 87.81, while the F-measure after coreference is deployed was 93.64. To evaluate the impact of coreference on the rest of IE system, we used as baseline the system with the entity and event coreference modules disabled, and compared the baseline against a system with entity coreference enabled, and against a system with all coreference modules enabled. Figure 5 plots CICERO’s performance for the MUC-6 domain and the three configurations, when the number of meta-rules increases from 1 to 26. Figure 5 shows two important ideas: first, it indicates that coreference almost doubles the performance of an IE system, a radical result different from previous IE system performance analysis reports. As Figure 5 indicates, CICERO without coreference barely approaches the 50% F-measure range, while with coreference enabled it reaches the upper 70% range.

<sup>3</sup>Throughout this paper we use MUC-6 dry run documents as training data, when not mentioned otherwise

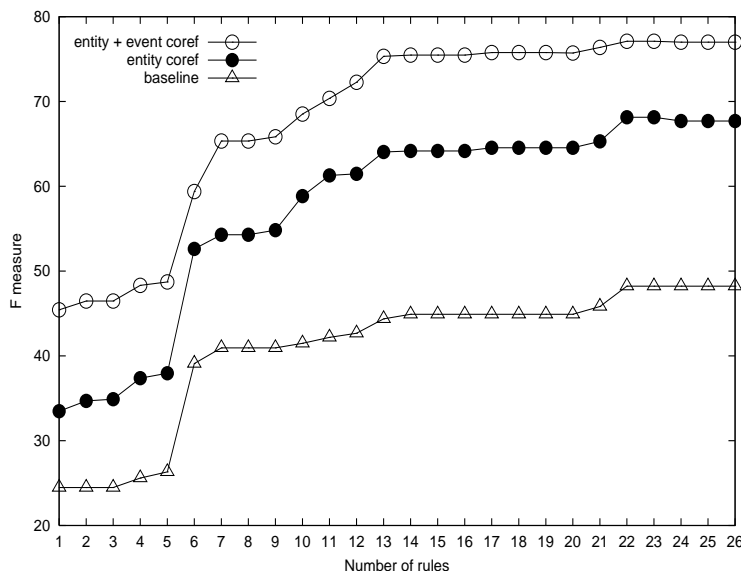


Figure 2: The impact of entity and event coreference on the performance of the CICERO IE system.

- The usage of *non-deterministic finite state automata (FSA)*, which can handle ambiguous input and can generate ambiguous output is vital for the handling of natural language ambiguities. The non-deterministic FSAs are used to postpone the decision taking for ambiguous output until sufficient contextual information (i.e. coreference chains or domain patterns) exists to eliminate ambiguities. For example, our named entity recognizer is able to associate multiple semantic tags to named entity, leaving the task of tag disambiguation to the named-entity coreference module, which has access to increased context. Our novel non-deterministic pattern matching technique reduces the propagation of errors between IE modules. In other words, we maximize the *recall* of the initial FSA-based modules (i.e. preprocessor, named-entity recognizer and parser) and use the domain-event recognizer and the coreference modules to increase *precision* due to their better understanding of the processed text.
- Non-deterministic FSAs need to be controlled* to avoid the generation of spurious input objects for the next processing stage. This observation is implemented in CICERO with a series of additional modules used to filter spurious objects. For example the “Partial POS disambiguation” module is used to control the phrasal parser output. If POS disambiguation is not enabled, the phrasal parser generates different syntactic phrases for each POS tag combination that matches a grammar rule. For example, the word “seek” can be both a noun and a verb phrase if additional constraints are not present in the text. Allowing the phrasal parser to generate all possible phrases complicates significantly the task of the entity coreference module, which attempts to build coreference chains for all noun phrases. That is why in the current CICERO implementation, we have added the POS disambiguation stage, which uses a voting algorithm between our internal lexicon and Eric Brill’s POS tagger to reduce the number of noun phrases constructed by the parser. Similarly, named-entity coreference can be regarded as a filtering stage for the spurious entries generated by the named-entity module.

### 3. THE ROLE OF COREFERENCE IN INFORMATION EXTRACTION

This section describes the novel coreference applications implemented in the CICERO information extraction system. We found the following five situations where coreference is useful. The first three situations are handled by the named-entity coreference module, and the last two by the open-domain coreference and the domain coreference correspondingly.

#### 1] Disambiguate incomplete named-entities

This task disambiguates entities that are likely to be part of a name, but can not be identified as such due to incomplete information. For example, the string “Language Computer” by itself can not be identified as a named entity, but it can be safely associated with an organization name if the string “Language Computer Corp.” appears as a coreference candidate in the same context.

#### 2] Disambiguate ambiguous named-entities

This task disambiguates entities which have either ambiguous length or ambiguous type. For example, the type of the named entity “Michigan” will be changed from location to organization if the string “Michigan Corp.” appears in the same context. The second example addresses the disambiguation of named entities with ambiguous length: if no NE pattern is matched, the string “Young’s” may correspond either to the organization named entity “Young’s” or to another named entity of shorter length: “Young”. Our solution to the above problem is to construct both entities in the named-entity recognizer. In the named-entity coreference module we maintain only the entity that is consistent with its context. For example, if the context surrounding the ambiguous entities contains the organization “Young’s Corp.”, the shorter length entity “Young” is deleted and “Young’s” is disambiguated to an organization named entity.

#### 3] Recognize of headline named-entities

The headline named entity recognition is considered to be a more complicated problem than the recognition of named entities located in the message body, because headline words are typically all capitalized or start with capitalized letters and case is a very important

feature in named entity grammars. Instead of developing a case-insensitive grammar, we propose a coreference-based approach for the headline NE recognition. We initially delay the processing of headline named entities until after the message body is processed. Then we use a longest match approach to match the headline sequence of tokens against entity names recognized in the first message paragraph. For example, in the headline “McDermott Completes Sale” only the string “McDermott” is disambiguated to an organization named-entity because “McDermott International Inc.” appears in the first message paragraph. Note that our case sensitive grammar would have matched the whole headline as a potential named-entity. By using coreference we are able to achieve better results with a simpler approach.

#### 4] Disambiguate nominal/prenominal references

The disambiguation of nominal and prenominal references increases significantly the recall of the pattern recognition mechanism implemented in the domain event recognizer. For example, if a domain pattern expects to see an organization name in a certain position, it will not match over the “it” pronoun, unless the pronoun refers to an organization name. The coreference algorithm is implemented in two steps: a *candidate retrieval* step which identifies possible candidates for the current coreference link, and a *semantic consistency filtering* step, which selects the first candidate that is semantically consistent with the anaphor. For semantic filtering we employ dedicated filters for pronoun, person name, organization name, and common noun candidates. For example, a common noun candidate is considered consistent with the anaphor if the two entities have consistent numbers and genders and the anaphor is a hypernym or synonym of the candidate in WordNet.

#### 5] Fill in missing template/templette slots

Following the domain event recognizer the domain coreference fills the template or templette slots that could not be filled by the previous module using FSA patterns. Due to the inherent complexity of the natural language, there are many situations when relevant information is not expressed in a form that can be captured by a reasonable pattern, hence the importance of domain coreference. The following example from a “natural disaster” message illustrates the problem:

... Although warmer temperatures have helped out Montreal today, and ice has come crashing down nearly everywhere, public utilities here are still a mess. Overnight, massive power outages knocked out a water filtration plant, ...

For the above text, CICERO’s domain event recognizer partially matches the <DISASTER DESTROYS ARTIFACT> pattern over the text “knocked out a water filtration plant”. Due to the incomplete match, the constructed templette lacks the type of disaster that caused the event, and the location and date of the event, which, even though present in the text, can not be extracted with an acceptable pattern. The domain-dependent coreference module is used in such situations to search the pattern context for entities that match the missing slots. In the above example, using domain coreference we are able to retrieve the disaster type (“ice”), the event location (“Montreal”) and the event date (“today”). The domain coreference uses the same candidate retrieval strategy as the previous open-domain coreference module, but each semantic filter is adapted for the type of entity that is compatible with the corresponding missing slot.

## 4. NON-DETERMINISTIC FSA

The addition of non-deterministic FSAs is another contributing factor in CICERO’s performance. The following examples show how non-deterministic FSAs are used to accommodate the natural language ambiguities. For example, in the text:

“... the storm hit Lincoln ...”

the named-entity recognizer generates both location and person named-entities for the string “Lincoln” because it does not have sufficient information to disambiguate between the two. This ambiguity propagates through CICERO’s modules up to the domain event recognizer which disambiguates “Lincoln” as a location named-entity with the pattern <DISASTER HAPPENED-IN LOCATION>. Another example was presented in the previous section, where coreference is used to disambiguate ambiguous named-entities. If non-deterministic FSAs were not used, the construction of ambiguous output, either of different type like “Michigan” as both location and organization name, or of different length like “Young” and “Young’s” would not be possible. The last example presented shows the application of non-deterministic FSAs to parsing. One of the problems we encountered during the implementation of the deterministic parser was the treatment of “that”, which can be both a determiner and a particle. The correct parsing of “that” is extremely important because it affects all domain patterns that contain nouns with this determiner, and also all the domain patterns expressed in relative form. For example, the pattern <LOCATION CONTAINED DISASTER> can not be matched over the text:

“... the flood that Florida saw today ...”

if “that” is parsed as a determiner part of the “that Florida” noun group. Our relative object pattern macro matches over forms like <OBJECT THAT SUBJECT VERB>, hence the recognition of “that” as a particle is crucial. Because the distinction between determiner and particle is extremely difficult for a FSA-grammar parser, we allowed both forms of “that” to be generated with equal priority.

In order to implement non-determinism in CICERO we had to provide solutions for the following issues:

#### Allow ambiguous input

Because CICERO’s backbone is implemented as a sequence of FSAs, one module’s ambiguous output becomes the next module’s ambiguous input. For example, if the named-entity recognizer could not decide between an organization or a person named-entity, both entities are passed as ambiguous input to the next FSA module, the parser. The solution we adopted in CICERO was to implement the interface between FSAs as a lattice of objects instead of a linear vector. Our lattice implementation allows for multiple objects to start at the same position, and for objects that start at the same position to end at different positions. The search algorithms were modified to scan all alternative lattice entries.

#### Allow the generation of ambiguous output

CICERO’s automata and search algorithm are implemented such that the first match obtained is always the longest match, which is preferred most of the time. The generation of ambiguous output is implemented by allowing the search process to continue after the first match, in one of the following two conditions:

- The solution path contains OR alternatives. In this situations, CICERO’s runtime search system investigates all the other alternatives in the OR constructs.
- There are grammar rules with equal priority as the rules used to obtain the first solution path. CICERO’s grammar rules are

prioritized and rules with higher priority are always preferred over rules with lower priority. Ambiguities are allowed in the search process by investigating all rules with similar priority as the rules used to obtain the first solution.

### Optimize the search on non-deterministic FSAs

The main problem we encountered in the implementation of searching non-deterministic FSAs was the high backtracking cost. In CICERO, each grammar rule can be associated with a set of actions, typically used to construct the corresponding output objects. The execution of rule actions during the search process has not only a high cost but is also useless considering that actions should be committed only when a search solution is found. Our solution to this problem is to implement the search algorithm as a *two-step process*: the first phase performs a graph search on the grammar FSAs. This step is a read-only process: solution search stacks containing all visited automaton nodes are constructed, but no actions are executed. The second phase of the search algorithm traverses the search stacks and executes all actions encountered. This approach provides the fastest search algorithm because it guarantees that only the actions associated with solution paths are executed.

## 5. PERFORMING OPEN-DOMAIN IE

A key issue that simplifies CICERO’s porting to new domains is the fact that CICERO cleanly separates the rule construction into domain-independent and domain-dependent parts. CICERO facilitates this separation with *meta rules*. A meta rule expands upon domain-independent *syntactic patterns* with domain-dependent *semantic constraints*. For clarification Figure ?? shows the implementation of a domain Subject-Verb-Object (SVO) pattern as an expansion of domain-independent meta rules. Note that while SVO patterns have by far the biggest contribution of all patterns in CICERO (i.e. SVO patterns plus coreference account for over 96% CICERO’s score for the MUC-6 domain), meta rules are not limited to SVO patterns: the same concept is used for the implementation of complex noun groups. Figure ?? shows the implementation of the “REMOVE” pattern, which matches over text constructs such as:

... Bluh Co. removed John Doe from the post of president...

or

... John Doe was fired by Bluh Co. from the position of CEO...

What is significant in this example is that the domain-dependent part of the rule is contained in the “with { ... }” construct, which defines the semantic constraints to be applied on the elements part of the SVO pattern: subject, verb, and object. The domain-independent meta rules, shown within the “expand { ... }” construct define the way SVO patterns can be expressed in the current development language (in this case English). We currently have implemented 35 meta rules that implement domain-independent syntactic constructs for the English language, for example the active SVO pattern, where the subject precedes the verb in active form.

The previous example indicates that there are at least two relevant things that have to be learned for a new domain: first and most importantly, one needs to detect the semantic constraints to be imposed on meta rules (i.e. the second part of Figure ??). Second, each syntactic pattern must be expanded with other applicable meta rules to increase the pattern coverage. It is however important to keep the meta rules associated with each syntactic pattern

```

expand {
  ActiveBase
  ActiveRelativeSubj
  ActiveRelativeObj
  PassiveBase
  ...
} with {
  ??label = "REMOVE"
  ??subj = in.B("isCompany")
  ??head = $REMOVE_WORD
  ??obj = in.B("isPerson")
  ??prep1 = "as" | "from"
  ??pobj1 = in.B("isPosition")
  ??semantics = makeRemove(...)
}

```

Figure 3: Domain pattern implemented as an expansion of domain-independent meta rules.

to a minimum, because too many meta rules may not only introduce spurious entries due to incorrect matches, but also slow down the system significantly due to the exponential number of patterns generated.

The open-domain IE platform implemented in CICERO allows the implementation of supervised machine-learning mechanisms capable of learning new rules for new domains, given a set of training texts and templates. The meta-rule framework and advanced usage of coreference algorithms guarantee large coverage of the learned rules. Figure ?? shows the infrastructure of such a rule learning algorithm. For the clarity of the presentation we describe the algorithm behavior on a simple example.

The algorithm input is two-fold: the first item is CICERO’s lattice after the the full parsing and the entity coreference steps. The lattice consists of a sequence of possibly overlapping noun, verb, and particle groups. Noun and verb groups may be tagged with semantic information, such as named-entity tags or WordNet information pre-tagged as domain-relevant (e.g. nouns such as “flood” and “tornado” are relevant for the “natural disasters” domain). The second input item for this algorithm step is the sequence of training templates. Consider the following training text “natural disasters” domain as example:

... flooding has become a way of life in Guerneville in the last several years. This gas station along the Russian River gets hit nearly every time. ...

and the corresponding pre-tagged template:

```

<NATURAL_DISASTER> :=
  DISASTER: "flooding"
  AMOUNT_DAMAGE: "this gas station"
  LOCATION: "Russian River"

```

During the first algorithm step all possible SVO patterns are generated and matched over the training text. Some validation algorithm must be used to preserve only the patterns that have a positive contribution to the overall score. During this step the following SVO pattern will be generated:

```

expand {
  PassiveBase
} with{
  ??head = "hit"
  ??obj = has facility feature
  ??semantics =
    set amount damage = object;

```

```

    set location = event adjunct location (if any);ActiveBase
}
ActiveRelativeObject
...
PassiveBase
...
} with{
??head = "hit" | "strike" | ...
??obj = has facility feature
??semantics =
    set amount damage = object;
    disaster type = fill using coreference;
    set location = event adjunct location (if any);
}

```

The output of the first algorithm step is a set of simple rules, which in CICERO are implemented as syntactic pattern that expand a single meta rule. This is the step where most IE systems stop. It is however obvious to us that stopping here leaves the sparse data problem unaddressed. The remaining two algorithm steps address this specific issue, first by increasing the coverage of the learned patterns, and then by using event coreference to include information that cannot be retrieved using pattern matching.

The second algorithm step expands the rules learned in the first step with additional meta-rules, and semantically equivalent words, such as WordNet synonyms. This step expands the rule generated in the previous example as follows:

```

expand {
  ActiveBase
  ActiveRelativeObject
  ...
  PassiveBase
  ...
} with{
??head = "hit" | "strike" | ...
??obj = has facility feature
??semantics =
    set amount damage = object;
    set location = event adjunct location (if any);
}

```

The above example shows how the previous rule is expanded by increasing the number of syntactic patterns covered. The rule coverage is further increased by expanding the head verb constraints with synonym verbs such as “strike”.

As our previous analysis of the CICERO IE system indicates, successful pattern matching rules are only half of the extraction process. Coreference, in the form of domain-independent entity coreference and domain-dependent event coreference, is responsible for the other half. Hence, a successful automatic domain customization algorithm must address the implementation of event coreference.

The task of event coreference is to fill in the slots missing in the templettes constructed by the previously learned meta-rules. Missing slots occur both due to incomplete matches and because most patterns are not designed anyway to fill all slots. It is important to note that not all missing slots should be filled using event coreference. Depending on the currently matched rule, some slots should be left empty because event coreference might introduce incorrect fills from the rule context.

We exemplify the use of the event coreference learning mechanism on the same example used throughout this section. When the “natural disaster” meta-rule learned in the previous step is applied on the training text it will fill in the slots corresponding to the artifact damaged (“this gas station”) and the disaster location (“Russian River”). Nevertheless, two important fields are missing: disaster type and disaster date. Filling the disaster type slot using coreference is beneficial because it successfully identifies “flooding” as the disaster type, but it fails on the disaster date slot. In this example, event coreference fills the event date slot with “last several years”, which is a spurious entry. Hence, the learning algorithm eventually marks the date slot in the templette generated by the above meta-rule as “not to be filled” by event coreference. The final learned templette will have the following form:

```

expand {

```

## 6. ACKNOWLEDGMENTS

This work was partially supported by the ARDA contract ARDA#2001\*H238400\*000 and by the “ARP:Knowledge Mining for Open-Domain Information Extraction” grant from the Advanced Research Program of the Texas Higher Education Coordinating Board.

## 7. REFERENCES

- [1] D. E. Appelt, J. R. Hobbs, J. Bear, D. Israel, M. Kameyama, A. Kehler, D. Martin, K. Myers, and M. Tyson. Description of the fastus system used for muc-6. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, pages 237–248. Morgan Kaufmann, 1995.
- [2] E. Brill. A simple rule-based part of speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, pages 152–155, 1992.
- [3] R. Grishman and R. Yangarber. Nyu: Description of the proteus/pet system as used for muc-7 st. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*. Morgan Kaufmann, 1998.
- [4] J. R. Hobbs, D. E. Appelt, J. Bear, D. Israel, M. Kameyama, M. Stickel, and M. Tyson. Fastus: A cascaded finite-state transducer for extracting information from natural-language text. *Finite State Language Processing*, page edited by Emmanuel Roche and Yves Schabes, MIT Press 1997.
- [5] G. Krupka. Sra: Description of the sra system used for muc-6. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*. Morgan Kaufmann, 1995.
- [6] W. Lehnert, C. Cardie, D. Fisher, E. Riloff, , and R. Williams. Description of the circus system as used for muc-3. In *Proceedings of the Third Message Understanding Conference (MUC-3)*, pages 223–233. Morgan Kaufmann, 1991.